National Research Council Canada
Conseil national de recherches Canada

Division of Electrical Engineering
Division de génie électrique

ERB-1012

A System for Calibrating the Wrist-mounted Laser Range Finder

S. Amid, B. Trethewey, and C. Archibald

November 1988

Canada

NRCC No. 29799

89-03321

Copyright 1988 by
National Research Council of Canada

Permission is granted to quote short excerpts and to reproduce figures and tables from this report, provided that the source of such material is fully acknowledged.

Copyright 1988 par
Conseil national de recherches du Canada

Il est permis de citer de courts extraits et de reproduire des figures ou tableaux du présent rapport, à condition d'en identifier clairement la source.

Additional copies are available free of charge from:

Editorial Office, Room 301
Division of Electrical Engineering
National Research Council of Canada
Ottawa, Ontario, Canada
K1A 0R8

Des exemplaires supplémentaires gratuits peuvent être obtenus à l'adresse suivante:

Bureau des publications, Pièce 301
Division de génie électrique
Conseil national de recherches du Canada
Ottawa (Ontario) Canada
K1A 0R8

A SYSTEM FOR CALIBRATING THE WRIST-MOUNTED LASER RANGE FINDER

S. Amid, B. Trethewey, and C. Archibald

ERB-1012
NRCC No. 29799

NOVEMBER 1988

TABLE OF CONTENTS

Page

ABSTRACT/RÉSUMÉ v
1. INTRODUCTION 1
2. LINEAR INTERPOLATION FOR CALIBRATION 2
2.1 Advantages of linear interpolation 2
2.2 Determining position in *X* 3
2.3 Interpolating in *Z* 6
3. ACCURACY DETERMINATION 8
4. CONCLUSIONS 10
ACKNOWLEDGEMENTS 10
REFERENCES 11
APPENDIX A Operating instructions for calibration apparatus and routines 13
A1. CALIBRATION APPARATUS 13
A1.1 Interface to the calibration bench 14
A1.1.1 Interactive mode 14
A1.1.2 Automatic bench movement 16
A2. DATA COLLECTION 16
A2.1 Collection control 16
A2.1.1 Hardware configuration 17
A2.1.2 Program interaction 17
A2.1.2.1 Intensity data collection 17
A2. 1.2.2 Range data collection 19
A2.1.3 Program hot keys 19
A2.2 Manual validation of intensity data 20
REFERENCES 21
APPENDIX B Software used to control the calibration bench 23
APPENDIX C Software used to collect and verify raw data 29
APPENDIX D Software used to calculate the *X* equations 51
APPENDIX E Software used in the generation of the Z lookup table 55

List of Figures

1. The mechanical arrangement of the wrist-mounted laser range finder 1
2. The range values collected at each *Z* 3
3. The distorted polar coordinate system of the range finder 4
4. A sample plot of intensity data with detected targets indicated 4
5. Calculated equations of the *X* contours 5
6. The detected and interpolated points on two range contours 7
7. A sparse plot of the Z LUT 7
8. Accuracy determination by examining calibrated range data 8
9. Calibrated range data from standoffs of 20, 40, 60, and 80 cm 9
A1. The vertical bench used in the calibration process 13
A2. A sample of the target used in the collection of intensity data 18
A3. The detected target positions used to calculate the *X* equations 20

ABSTRACT

The wrist-mounted laser range finder developed in this laboratory provides depth information which is distorted due to the scanning mechanisms and the perspective distortion present in all range finding devices. For the wrist-mounted scanner in particular, this problem is pernicious because of the large depth of field for which this sensor is designed. Methods of removing the nonlinearities to provide an orthographic projection have been developed for other range finders, but for a variety of reasons, these have been found unsatisfactory. A new method, apparatus, and procedure for calibration are described in this paper. Linear interpolation over multiple known samples of uncalibrated data are used to build a lookup table. The lookup table, stored in RAM, is accessed in realtime using the scanner hardware to provide calibrated data at a rate of 15 calibrated scans per second. One of the biggest advantages of the procedure that is described is that it is very easy to use and relatively fast. The entire calibration is accomplished in a few hours sitting at a personal computer.

RÉSUMÉ

Le télémètre à laser monté sur le poignet mis au point au laboratoire fournit des données de profondeur qui sont biaisées à cause des mécanismes de balayage et de la déformation de la perspective qu'on retrouve dans tous les dispositifs télémétriques. Dans le cas particulier du télémètre à laser monté sur le poignet, ce problème est pernicieux à cause de la grand profondeur de champ pour laquelle ce capteur a été conçu. Des méthodes ont été mises au point pour éliminer les non-linéarités dans d'autres télémètres de façon à réaliser une projection orthographique, mais elle se sont avérées insatisfaisantes pour diverses raisons. Le présent document décrit une nouvelle méthode, un nouvel appareillage et un nouveau mode opératoire pour l'étalonnage des télémètres. Des tables de renvoi sont établies par interpolation linéaire de plusieurs échantillons de données non étalonnées. Ces tables, mises dans une mémoire à accès sélectif (RAM), sont consultées en temps réel au moyen système à balayage pour étalonner les données à un rythme de 15 profils étalonnés par seconde. Un des plus grandes avantages de la méthode, décrit dans le présent document, est sa facilité d'utilisation et sa rapidité relative. L'étalonnage est entièrement réalisé en quelques heures depuis un ordinateur personnel.

A SYSTEM FOR CALIBRATING THE WRIST-MOUNTED LASER RANGE FINDER

S. Amid, B. Trethewey, and C. Archibald

1. INTRODUCTION

The problem of calibrating the 3-D laser range finder has been an important issue in the application of sensor technology in the Laboratory for Intelligent Systems of the Division of Electrical Engineering. A laser range finder that was specifically designed to be mounted on a robot wrist is described in ref. (1). This particular range finder was designed to have a large depth of field. The mechanical arrangement of the range finder is shown in Fig. 1. It is assumed that the reader is familiar with the functionality of this sensor. Several methods have been used to transform the data from range finders to real world Euclidian space (2, 3). This paper deals specifically with calibrating the wrist-mounted laser range finder using linear interpolation.

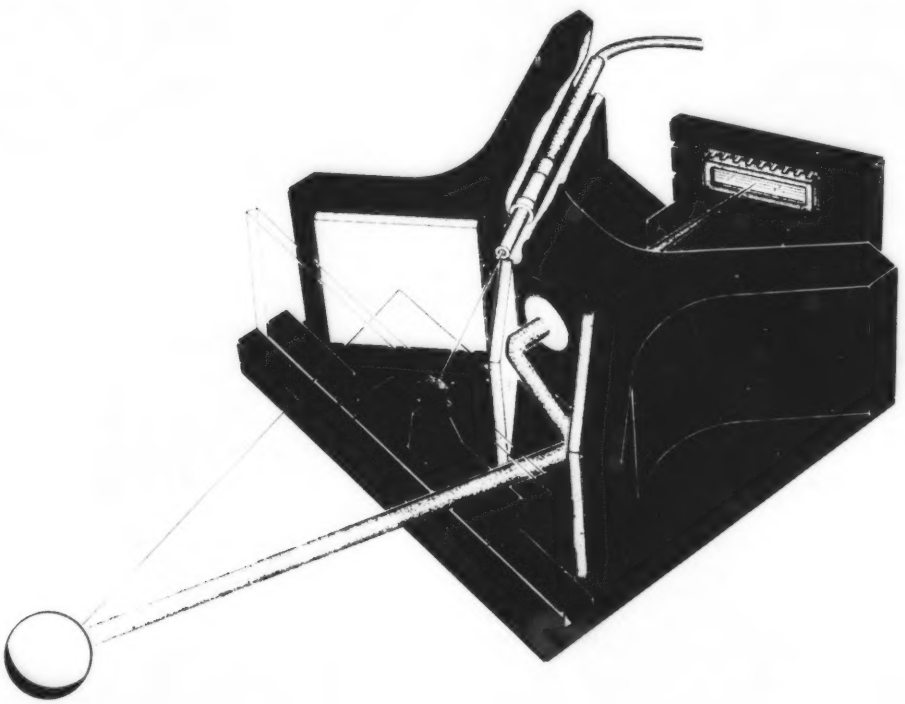

Figure 1. The mechanical arrangement of the wrist-mounted laser range finder to be calibrated.

The objective herein is to document the calibration method and describe the apparatus as well as the use of the system that has been developed. In Section 2 the building of the calibration lookup table (LUT) using linear interpolation is described. The success of the calibration process can be measured by the ability of the range finder to produce data in Cartesian coordinates that accurately depict the surface being scanned. The accuracy of the scanner after calibration is discussed in Section 3. The conclusions on the methods described can be found in Section 4.

As described in Appendix A, Sections A1 and A2, the apparatus for collecting the raw data to be used in the calibration and the procedure to do that collection are nontrivial. These aspects of the calibration are theoretically not crucial to the calibration itself but if not implemented in a user friendly manner will render the entire process cumbersome and unusable.

The Appendices B, C, D, and E contain the code which is used in the calibration process for the reader who is interested in detail at that level.

2. LINEAR INTERPOLATION FOR CALIBRATION

The calibration process described here addresses problems of geometric and radiometric distortion related to optical and electrical components used in the range finder (1).

In Appendix A, Section A2, the collection of raw range data and intensity data at known standoff locations is described. The remaining task is to couple the distorted polar coordinate system and the more useful Cartesian system by creating a lookup table. The lookup table contains Cartesian coordinate locations and is indexed by the raw range data and the *azimuth* position. This azimuth position is simply the index, 1–255, of the scan drive system, which represents the angle of projection of the laser beam from the scanner. We interpolate between two adjacent scans of range values for each azimuth to obtain the Z values in the Cartesian coordinate system. These Z values are placed in the LUT. The X values for the X–Z coordinate pairs are computed using the Z values obtained from the table and the linear equation of the azimuth in question. The method used to calculate the X equations is described in Section 2.2.

2.1 Advantages of Linear Interpolation

Linear interpolation was used for calibration for a variety of reasons. Previous methods involved the use of spline interpolation in the calibration process (2). The results obtained from these methods were not satisfactory because of the long field of view in this particular range finder.

The calibration lookup table is affected by any degradation in the quality of raw data. There is some degradation in the raw data at the extremity of the scan due to partial blockage of the return path of the laser in this particular range finder. This problem is not serious, as it is localized to small regions at one end of the scans at short standoffs and at the opposite end of the scan at large standoffs. Linear interpolation is a local function, which will restrict this phenomena to a small number of points in the extremities of the lookup table.

Owing to the geometric design of the camera, the resolution of this sensor is much higher at short standoffs than at larger standoffs where the fine resolution is less important.

This function can be represented by showing the change in range values collected from the range finder as it is moved through its field of view. Figure 2 shows Z vs. range values. Note that at short standoffs, the range values are changing very fast compared to the change in Z. As is discussed in Section 3, this allows about 20 times greater accuracy at the short standoffs than at a standoff of 1 m.

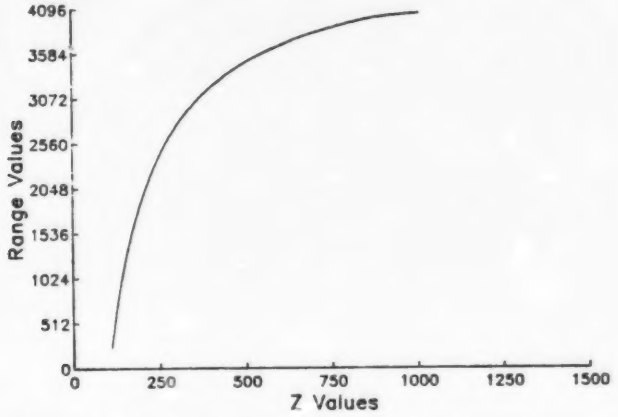

Figure 2. The range values collected at each Z. This shows that the shorter standoff distances have much higher resolution in range than the larger standoffs. The range values at the centre of the scan are used for this plot.

2.2 Determining Position in *X*

Calibration for the *X* direction is done by computing the linear equations of the data passing through the azimuth contours (see Fig. 3). This is possible because the azimuth contours in the range–azimuth coordinate system are colinear to the *X* contours in the *X*–*Z* coordinate system. This is evident if the reader considers that the shape of the field of view will not change as the data are transformed to Cartesian coordinates. Once the linear equations of the azimuth contours are calculated, these can be stored and used to calculate the *X* coordinates by simply substituting the *Z* values into the equations as they are extracted from the LUT.

The method used to calculate the equations of the azimuth contours is as follows. Several range–azimuth coordinate pairs must be obtained for each azimuth. A line is fitted to these coordinates which minimizes the error of the equation in a least-squares sense. The resulting equations can be used as the equation of the *X* contour, since the *X* contours are colinear to the azimuth contours. Determining the range–azimuth coordinate pairs at each azimuth is done experimentally. Intensity data are collected over targets which consist of a number of equally spaced black bars. An example of the raw intensity data collected from a target is shown in Fig. 4. The larger gap observed in the middle corresponds to the spacing between the bars at the centre which is twice the size of others to indicate the *X* origin of the Cartesian coordinate system. The targets used have differing bar widths to allow for the changing resolution of the camera through its field of view.

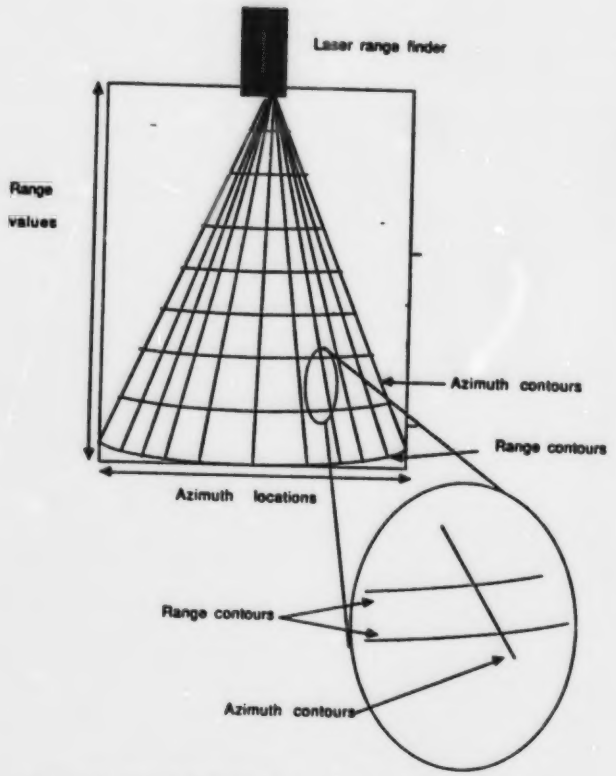

Figure 3. The full field of view of the range finder. The distorted polar coordinate system is illustrated.

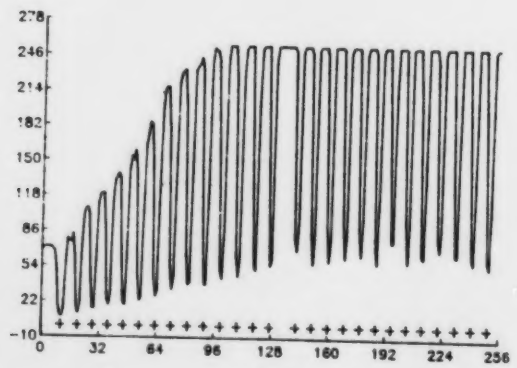

Figure 4. A sample plot of intensity data. The detected targets are indicated.

Using these black targets, the *X* values are computed for each azimuth in the following manner. The intensity data collected are averaged over 50 pixels to normalize the peaks created by the black bars with respect to the background. The zero crossings are detected using the first and second derivatives and a Sobel operator. The first derivative is used to detect a positive change in sign to indicate the zero crossing. To validate this zero crossing the second derivative must be larger than a threshold value. The azimuths of the detected targets are calculated to one quarter of an azimuth step at the zero crossing points.

The *X* origin of the Cartesian coordinate system which is at the centre of the large spacing between the bars is detected. From this position, values of *X* equal to the distance between two bars are assigned to each azimuth position previously calculated. These measured target azimuth values and their corresponding *X* values are used to linearly interpolate *X* values for each azimuth step between them.

On each scan the first and the last target detected rarely correspond to the ends of the scan. Therefore, the first and the last positions detected are used as the starting points for extrapolation to the ends. Thus, the equations of the azimuths at the outer edges of the field of view are not as accurate as the central region.

The *X* values have been calculated for each of the 255 azimuths. These values are written to a file using the convention described in Appendix A, Section A2.1.2.1 for later input to XEQNS. All of the code for this data collection can be found in Appendix B.

The equations of the *X* contours are calculated as described above. A sparse plot of these *X* contours is included as Fig. 5. This figure gives the reader a feeling for the shape of the field of view of the range finder as well as an indication of how the resolution in *X* is decreasing as the standoff increases. The code for the calculation of the *X* equations can be found in Appendix D.

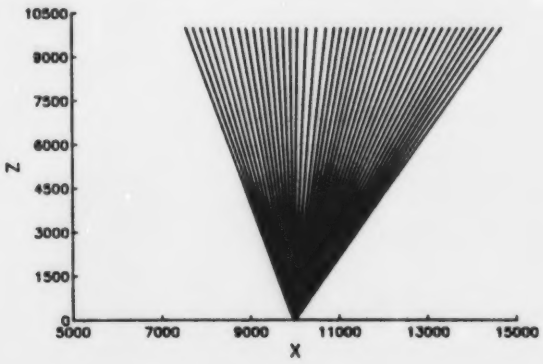

Figure 5. Calculated equations of the *X* contours.

2.3 Interpolating in Z

The data collection described in Appendix A, Sections A2.1.2.1 and A2.1.2.2 illustrates how points on a plane at different Z positions in the Cartesian coordinate system are collected. The field of view of the range finder is illustrated in Fig. 3. The concentric curves in this figure represent all points with radius equal to a range value. For ease of reference, a collection of these points for a given range value is termed a *range contour*. The CCD, through the peak detection algorithm (4), can resolve 4096 different range contours representing 12 bits of information. The scanning mechanism scans with 255 azimuth steps in the *X* direction. For ease of reference, a collection of all range readings (4096 values) at the same azimuth position is termed an *azimuth contour*. These azimuth contours are radial lines extending outward from the range finder over the full field of view (Fig. 3).

The range and azimuth contours discussed above represent the distorted polar coordinate system of the range finder. The mapping of this system to the Cartesian coordinate system is achieved through sampling the field of view of the range finder with Z collection planes. The frequency of sampling is directly related to the accuracy needed for the calibration table. (For a discussion on accuracy refer to Section 3.) The Z collection planes intersect all azimuth contours as well as a number of range contours. Consequently, information can be collected from both coordinate systems about every point on these planes. These points establish a direct relationship between the distorted polar coordinate system and the Cartesian coordinate system from which the Z lookup table can be constructed.

Currently for a calibration over the range of 11 cm to 1 m, 567 files of raw range data were collected, each of 1 Kbyte in size. The density of the raw data collection changes as the standoff increases. This is in keeping with the varying resolution of the scanner. The raw data collection densities are as follows:

Range (mm)	Density (mm)
111 - 200	0.5
200 - 300	1.0
300 - 600	2.0
600 - 1002	3.0

We believe that this is more dense than required, but the data collection routines are automated to the degree that makes this process convenient, in fact, the operator need not be present for most of the time the data collection is in progress and there is no significant computational penalty for having too much data. However, if the interpolation is too coarse, i.e., the raw data collected are too widely spaced, the accuracy required may not be attained.

The range value and azimuth position (in the polar coordinate system) are collected from the scanner hardware for each point within the Z collection plane. The Z position, in Cartesian coordinates, is recorded from the calibration bench. The interpolation process involves choosing points for which pertinent information in both coordinate systems is known. These points, illustrated as detected points in Fig. 6, occur at the intersection of the azimuth contour, the range contour, and the Z collection plane. Two such points are needed for the outer limits of the interpolation regions. These points must be located on adjacent Z collection planes as well as belong to the same azimuth contour. These constraints effectively define the range as the independent variable in the polar coordinate system. It can be assumed that the *X* and Z axes in the Cartesian coordinate system are independent. Thus, linear interpolation can be applied to calculate Z values for any range value within the

interpolation region without regard to the *X* direction. Only values corresponding to the 11 bit range contours (2048 positions) within the interpolation region are calculated to restrict the lookup table to a manageable size. The lookup table is populated horizontally by repeating the process described above for all azimuth contours and is populated vertically by iterating between all collection planes within the working range of the scanner.

The Z values within the lookup table are rounded to the first decimal place. This corresponds to a resolution of 0.1 mm for the table entries. These calculated values are stored as integers in the Z table. They are indexed by their corresponding range and azimuth values. The table has 2048 × 256 entries and occupies 1 Mbyte of memory. The lookup table is inspected for gross errors by plotting a sampling of the table. In Fig. 7 the LUT is plotted sparsely, giving the reader an idea of the shape of the data in the table. Note that the LUT resembles an inverted polar coordinate system, which is what one would expect to remove the spherical distortion in the range data.

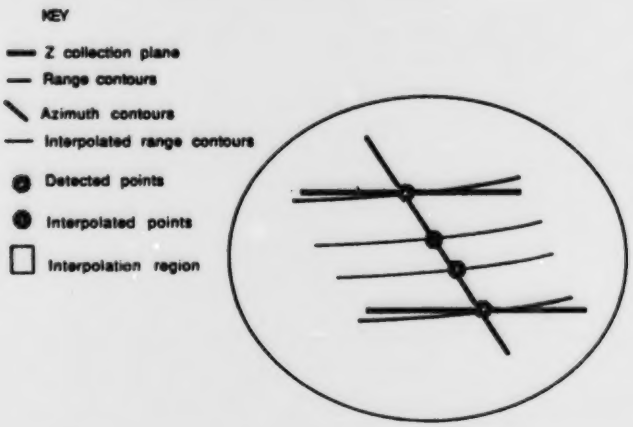

Figure 6. The detected and interpolated points. Range contours passing through known Z planes are shown, as well as two interpolated range contours.

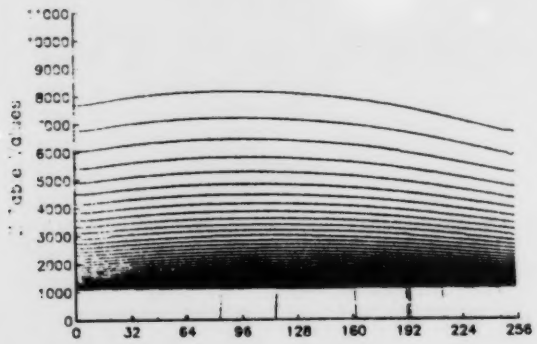

Figure 7. A sparse plot of the Z LUT.

3. ACCURACY DETERMINATION

Determining and expressing the accuracy of the profile scanner after calibration is difficult for several reasons. The sampled points are not equally spaced along the scan. The values that are returned after calibration lookup are *X*–*Z* pairs in the local coordinate system of the scanner. The calibration was designed to correct for the nonlinear geometry of the scanner. These nonlinearities result in the scan lines having *X* values that are more densely spaced at the centre of the scan than at the outside. The accuracy of the range data is non-linear in both *X* and *Z*.

For the present time the accuracy of the *Z* direction only is considered since this is more important to our current efforts. To do this calibrated data are collected from a flat surface, at several known distances from the scanner. Plotting and examining this data show

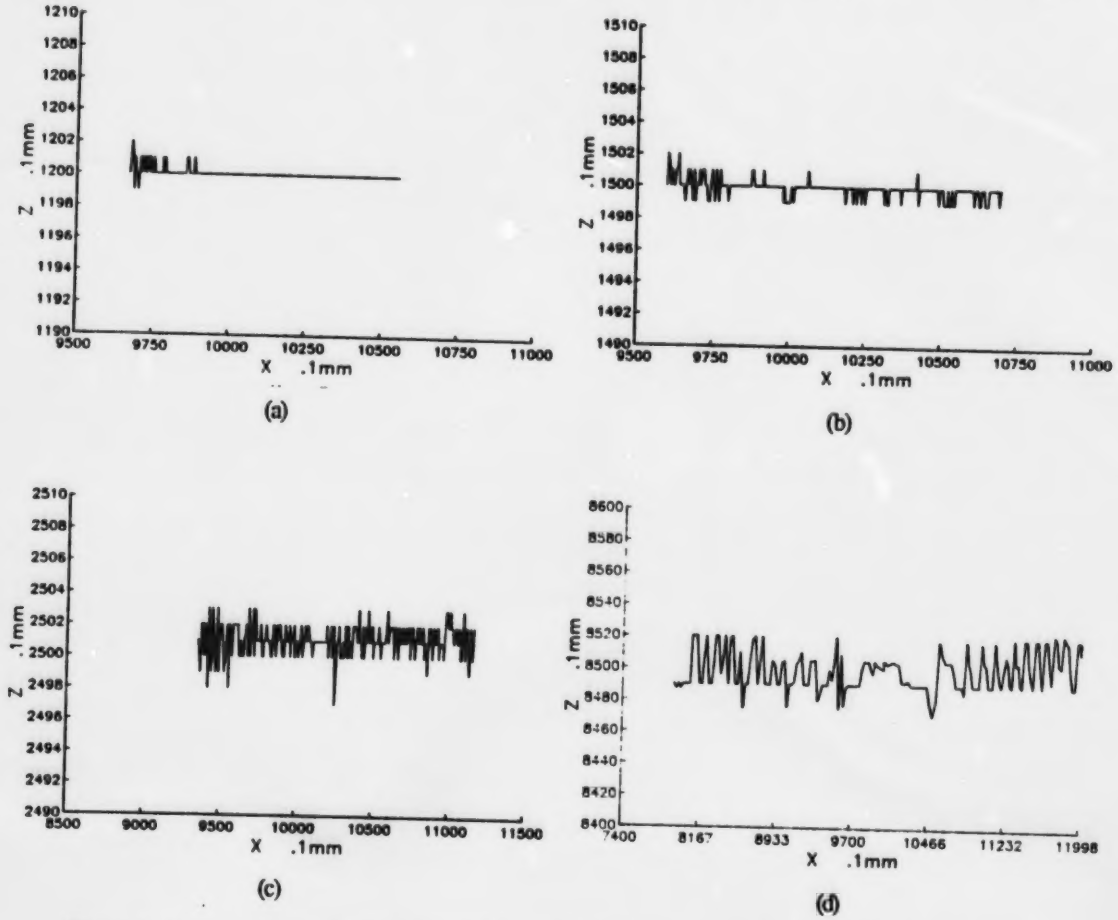

Figure 8. Accuracy determination by examining calibrated range data for a flat surface with a very high gain on the Z axis: (a) standoff = 12 cm, (b) standoff = 15 cm, (c) standoff = 25 cm, (d) standoff = 85 cm.

that the deviation from a straight line is comparable to the resolution in Z that is expected by the designers of the camera (1). The calibrated Z values deviate from a straight line by less than 0.1 mm at a standoff of 12 cm (Fig. 8a), deteriorating to ± 0.1 mm at 15 cm (Fig. 8b), ± 0.2 mm at 25 cm (Fig. 8c), and ± 2 mm at a standoff of 85 cm (Fig. 8d). To give an idea of the shape of the field of view and how the resolution changes with standoff Fig. 9 is included showing the same object viewed from several standoffs. The reader will note that at the very short standoffs the accuracy of this particular scanner suffers at one end of the scan. As the next generation of these scanners is currently being developed, the designers are becoming more aware of the cost/benefits of small geometrical modifications in the eventual calibrated performance.

Expressing the accuracy of a range finder that is measuring the distance to an ideal flat surface is somewhat different than expressing the accuracy, or ability of the sensor to perform some real world task. There are other issues to be examined before the applicability of the scanner can be determined. The shadow effect, for example, is greatly reduced using the synchronized scanning technique, but measuring this is truly dependent on the application. Usually a machine vision system must express its accuracy in terms of the particular application, for example, "these manufactured parts will be inspected to an accuracy of 1 mm in diameter."

Another major factor in expressing accuracy is the reflectivity of the surface being scanned. If the CCD saturates because of a highly reflective surface, either the accuracy will drop dramatically or no data at all will be available. This problem has been examined closely, and it is currently believed that the laser power can be dynamically attenuated to ensure that a reflective surface will not saturate the CCD. This attenuator is expensive, and it is not known if this will be a cost-effective approach.

The consistency of the intensity of the reflected laser power affects the accuracy of this sensor. It has been observed that by changing the intensity of the laser power the detected range will vary slightly. It is believed that this is caused by a change in shape of the peak that is read from the CCD and used to determine the range. Because the CCD in this

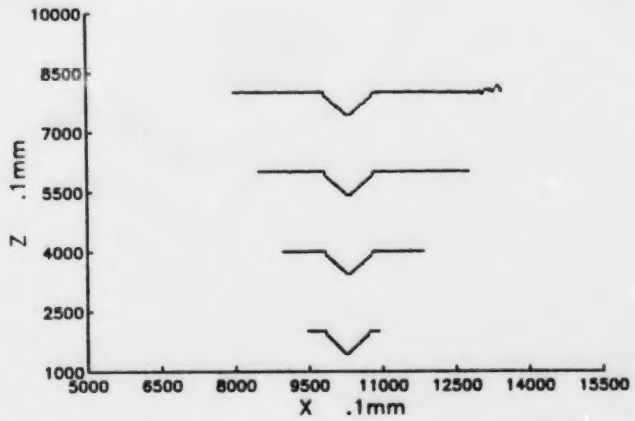

Figure 9. Calibrated range scans of a step pyramid from standoffs of 20, 40, 60, and 80 cm. The resolution decreases as the field of view widens with greater standoff distance.

device is inclined severely from the image plane (1) it appears that increased spot size on the CCD affects the behavior of the peak detector. Again, the addition of the attenuator is seen as a solution to this problem, and it does not affect our research progress at the present time.

Accuracy in the *X* direction is a slightly different issue. Because the linear interpolation depends on location of known targets for mapping the data into Cartesian coordinates in the *X* direction, the accuracy of the data at the extreme ends of the scans, approximately five azimuth steps on each end, is not expected to be as good as at the centre of the scan.

4. CONCLUSIONS

Calibration of the wrist-mounted laser range finder requires that large amounts of raw data be collected. The collection must be partially automated to make the entire procedure manageable. The data collection cannot be totally automated at this time because the intensity of the laser must be increased as the range finder moves farther away from the calibration target. Since every range finder is slightly different geometrically, each one must be calibrated individually. Also, when any modification to the range finder is made, it must be recalibrated.

The use of linear interpolation for calibration of the laser range finder has been successfully implemented on a personal computer. This method gives better results than the previously used B-spline method for this type of range finder. The errors at the extremities of the range profiles, which are inevitable at some standoffs, do not propagate errors to other parts of the lookup table. Linear interpolation localizes the errors in the table to the positions where the range finder cannot collect accurate data.

Accuracy to the the full 12-bit capability of the range finder is not achieved with the 2048×256 LUT. This can be improved by adding another step in the software that accesses the LUT. Currently the least significant bit is dropped to access the 11-bit table. This bit can be used to interpolate between the values in the LUT to give 12 bits of accuracy. This has not yet been done, as it is doubtful that this 12th bit, which is computed by interpolation in the peak detection hardware (4), will improve the accuracy of the range finder to better than the results reported here using 11 bits.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the efforts of Douglas Taylor, who was largely responsible for the design and construction of the calibration apparatus. He has also coded the target detection method for calibration in the *X* direction. Grant Bechthold and Michael Duggan have constructed the range finder and provided endless support and ideas. Nestor Burtnyk has provided guidance and many useful discussions. Evelyn Kidd made many valuable suggestions regarding the preparation of this text.

REFERENCES

[1] M. Rioux, G. Bechthold, M. Duggan, and D. Taylor. Design of a large depth of view 3-D camera for robot vision. Opt. Eng., 26(21): 1245–1250; 1984.

[2] F. Bumbaca, F. Blais, and M. Rioux. Real-time correction of 3-D nonlinearities for a laser range finder. Opt. Eng., 25(4): 561–565; 1986.

[3] L. Cournoyer. Procédure d'ajustement et de calibration pour capteur tri-dimensionnel à balayage synchronisé. NRC/ERB-998, National Research Council of Canada, Ottawa, Ont. January 1987.

[4] F. Blais and M. Rioux. Real-time numerical peak detector. Signal Process., 11: 145–155, 1986.

APPENDIX A

This section contains a detailed description of the calibration apparatus used, and operator's instructions on how the raw data collection has been arranged.

A1. CALIBRATION APPARATUS

The automation of data collection requires special equipment and software interfaces to control this equipment. In this section the automated calibration bench is described along with the interface designed to control it.

Calibration is entirely dependent on the accuracy of the data on which it is based. As such, the device used in the collection of data must be capable of moving the range finder to defined positions accurately and reliably. To satisfy these requirements, the vertical calibration bench shown in Fig. A1 was designed and constructed.

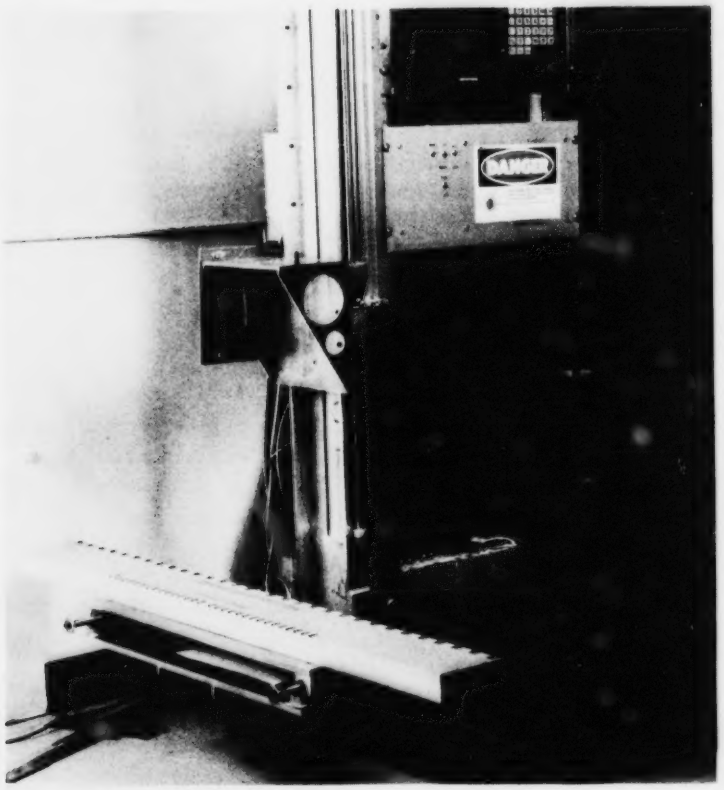

Figure A1. The vertical bench used in the calibration process.

This bench is a frame with a vertical slide mounted on one side. On this slide is mounted a platform for the attachment of the scanner which is being calibrated. The position of this platform is controlled by a lead screw which is driven by a stepper motor mounted at the top of the frame. The stepper motor is controlled by an intelligent positioning stepper motor controller which can be interfaced to a computer (A1).

In theory, the stepper motor should be capable of moving the platform to any given position with minimal error. In practice, this is not the case. Due to the internal characteristics of the motor, the buildup of momentum, and other influences, the motor does not always move with the expected precision. Thus, an independent sensor is required which can read the exact position of the bench (A2). From this position, the motor can be stepped up or down to compensate for these errors.

Measurement of the current position is provided by a digital readout of a graduated glass slide. The position is available to the host computer through the stepper motor controller. This measurement is accurate to within 0.005 mm. One step of the stepper motor will move the platform approximately 0.012 mm.

A1.1 Interface to the Calibration Bench

A general purpose interface to the bench, called BENCH, was developed for the PC/AT in C. This program has two major capabilities. First, it can be used as a simple way of interactively controlling the stepper motor controller. Second, based upon the setup of the bench, it can move the platform to a desired position with the best accuracy the bench is capable of producing. To use BENCH, the serial line from the bench must be connected to COM1 of the PC. The configuration of this port is performed by BENCH. The digital readout should be set to metric measurement. Refer to the code in Appendix B if more detail on this connection is required.

A1.1.1 Interactive Mode

To start BENCH in interactive communications with the bench, simply run it with no command arguments. There are several lines telling the user that it is in interactive mode and that hitting ESC will exit the program. Anything typed will be sent to the bench and anything that the bench sends back will be displayed. It should be noted that the bench is not user friendly. If a command is typed incorrectly, the bench may go into an unknown state and not accept any more commands. If this happens, the bench must be reset and the procedure tried again.

Four important sequences of commands which the user should be familiar with are described:

1. Bench Position Display

C[RETURN]
B[RETURN]

This sequence causes the stepper motor controller to read the current position of the bench from the digital readout. The position is then displayed in the format ***Z=XXXXX.XXX***. This value should match the value displayed on the readout. When

working in metric, the value returned will be in centimetres, while the readout will be in millimetres. The process of receiving the current position from the digital readout will cause the readout to beep once.

This sequence can be useful in testing the communications link to the bench.

2. Bench Initialization

I[RETURN]
F 10[RETURN]
S 25[RETURN]
R 230[RETURN]

This sequence initializes the stepper motor driver. The values given are values which were found through experimentation and calculation to provide the most efficient and quiet operation of the bench. This initialization should be performed whenever the bench has been turned off or reset.

3. Bench Movement (Relative)

To move the bench down:

–[RETURN]
N *nnnn*[RETURN]
G[RETURN]

To move the bench up:

+[RETURN]
N *nnnn*[RETURN]
G[RETURN]

These sequences will move the bench either up or down. The number ***nnnn*** is the distance to move. This distance is measured in two one-thousandths of an inch (***nnnn*** = 2000 = 1 in. = 25.4 mm). The distance can be from 0 to 65535 in size.

4. Bench Movement (Absolute)

To set the bench home position:

A[RETURN]

To move to an absolute position:

P *nnnn*[RETURN]

The first command is used to set the home position, from which all absolute positions are relative.

The second command will move the robot to a position relative to the last home position set. The value ***nnnn*** is a number between 0 and 65535 in units of two

one-thousandths of an inch. If the home position has not been set, the bench will move to an unknown location. If this happens, the stepper motor controller must be reset and the procedure begun again.

A1.1.2 Automatic Bench Movement

BENCH can be used to move the bench to an exact position. To do this, two command arguments are required:

BENCH *nnnn.nnn hhhh.hhh*

The first argument, ***nnnn.nnn***, is the position to which the bench is to be moved. The second, ***hhhh.hhh***, is the position of the bench at the last home position set. Both measures are in millimetres.

The program will then take control of the bench and move it in the following manner:

1. move to the approximate location
2. move towards the requested position until the position is passed
3. move back one step if the last position was closer

This algorithm ensures that the position of the bench will be as close to the desired position as possible.

The bench is moved according to the digital readout and, as such, the resulting position will have the maximum accuracy of the bench. Each time the bench moves, it will check the current position. This results in a beep from the digital readout.

Beware that failure to initialize the bench and set the home position before running BENCH in the automatic movement mode may cause erratic behaviour.

A2. DATA COLLECTION

The calibration process is based on the characteristics of an individual range finder. Each one is slightly different geometrically. As such, a large proportion of the time involved in calibration is spent in acquiring the necessary data for the range finder in question. Such collection can be tedious and prone to error if done manually, so the process was automated as much as possible. This makes the calibration data collection not only less time-consuming, but also more reliable, since there are fewer outside influences, and in particular, less chance of operator error. The C code for data collection can be found in Appendix C.

A2.1 Collection Control

Actual collection of the data for calibration is performed by COLLECT. This program was developed and written in C on the PC/AT.

A2.1.1 Hardware Configuration

COLLECT will call BENCH to move the calibration bench as required. As such, the serial line from the bench should be connected to COM1 of the PC.

COLLECT will also be accessing the range finder, via the 68000-based range finder computer. The scanner hardware should be loaded with the program CALIB.HEX and be running. This is the program that sends uncalibrated range profiles to the PC continuously. The parallel link from the scanner hardware to the PC should be connected.

Once the hardware has been set up properly, simply run COLLECT. The program does not have any arguments.

A2.1.2 Program Interaction

Upon starting, the first action of COLLECT is to call BENCH in interactive mode. In this mode, the bench can be initialized, moved, and set in the home position as explained in Section A1.1.1. This home position must be lower than any position at which data will be collected by at least 1 mm. If there is no change from a previous bench setup and the bench has not been reset or turned off, the setup need not be repeated. However, it is best to move the bench to the home position, since COLLECT will require the exact location of that position. Once the setup is complete, type ESC.

COLLECT will then request the location of the home position. Enter the location (in millimetres) displayed on the digital readout. Once this value has been entered, a menu will be displayed which has three options; collect intensity data, collect range data, or quit. The program then waits for the user to type a key corresponding to one of these choices. Intensity data are used in the generation of the *X* equations while range data are used in generating the *Z* table. Further information on the processing of this data is provided below.

All the files created by COLLECT are saved in the current directory.

A2.1.2.1 Intensity Data Collection

In collecting intensity data, the process is simply to move the bench to the proper location, scan the target for intensity (the amount of light reflected by the surface) changes, and from this scan detect the location of bars in the target. Since the bars are separated by a known distance, the world coordinates of each of the detected points is known, and from these points any remaining points can be interpolated. A portion of the target used for this process is shown in Fig. A2. The gap in the centre is used as a point of reference and is given the value zero for the *X* axis when interpolating.

Upon entering the intensity data collection portion of the program, COLLECT will request the following information:

lowest value of Z — this is the first location at which data are to be collected. This point should be higher than the home location. If it is not, all locations lower will have data collected at the current position of the bench.

highest value of Z — this is the maximum point of data collection. Data will not be collected any higher than this point. Because of the limitations of the stepper motor controller, this value must not be more than 832 mm above the home location.

step value of Z — this is the step distance between each point of data collection. Once a valid set of data has been collected at a given location, the bench will be moved up by this distance.

distance between targets — this is the distance between centres of the target bars used in the intensity data collection. The narrow targets are used for the high resolution, short standoff area of the field of view.

Once these values have been entered, the program will begin collection. At each bench position, it will read in a set of 10 scans, take their average, and attempt to find all of the targets. It will continue taking new sets of scan lines until it finds a set from which all of the targets can be derived.

The process of finding a target is based upon the fact that a target will show as a drop in intensity. If the targets are chosen such that the number of points scanned on a target is relatively small, this drop will exhibit a definite minimum, which can be located mathematically. The process used in this program is to take the first and second derivatives of the scan and look for a positive change in signs of the first derivative while the second derivative is greater than a threshold. This method will locate any drops in intensity which have a tight local minimum. If the minimum is too large, it becomes less dependable since the difference between adjacent points comes into the range of mechanical error.

Validation of a scan and its targets is based upon the fact that the the detected targets are roughly the same distance apart (except for the centre) even in uncalibrated data. As such, the ratio of distances between one pair of targets and the next pair should remain close to one. If this is not the case, then it is likely that a target has been missed or a target has been misplaced. The centre gap is handled as a special case. It should be pointed out that this method of validation does not detect missing end targets.

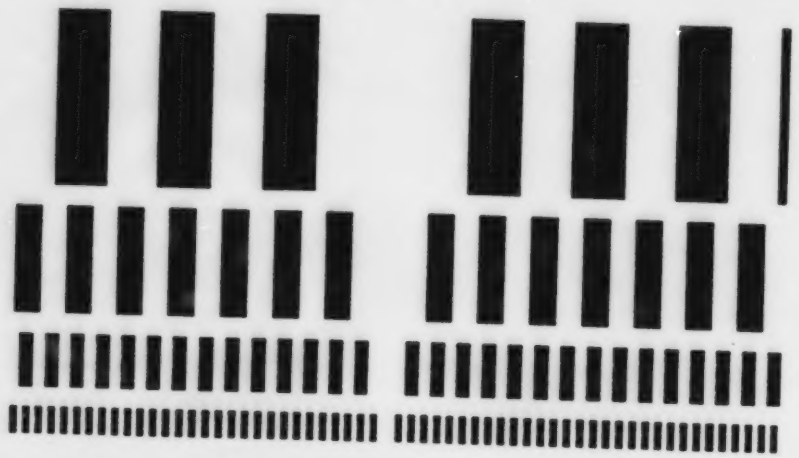

Figure A2. A sample of the target used in the collection of intensity data.

Upon successful validation of the scan, COLLECT will assign values to the target locations according to the *distance between targets* in the Cartesian coordinate system and interpolate the missing points of the 255 azimuth locations of the scan. These values will then be written in binary format to the file *XXXXXXXX*.XOP where *XXXXXXXX* is the truncated Z distance * 1000.

Two additional files are updated. The current Z position is appended in ASCII format to the file MANIFEST. This file is later used by XEQNS, as a reference to locate the data files. The current Z, position of the reference gap, number of targets located, and points which were assigned to targets are saved in the file X_Z.PTS in ASCII format. This file is later used by VALIDATE to plot all of the targets for manual validation of the target locations. Once the program has completed collection at all the requested bench positions, the opening menu is displayed again.

A2.1.2.2 Range Data Collection

Collection of range data is much the same as the collection of intensity data. The target scanned is a flat brushed metal surface designed to scatter the laser beam uniformly. The bench is moved to the proper location and an average of ten scans taken. Providing the scan is valid, the data are saved to a file named *XXXXXXXX*.ROP where *XXXXXXXX* is the truncated value Z × 1000.

The criterion for validation of the range data is that there must not be any substantial difference between adjacent points in the scan. Since the surface being scanned is flat, the raw range data will be a smooth curve.

Upon entering the range data collection portion of the program, COLLECT will request the following information:

lowest value of Z — this is the first location at which data are to be collected. This point should be higher than the home location. If it is not, all locations lower will have data collected at the current position of the bench.

highest value of Z — this is the maximum point of data collection. Data will not be collected any higher than this point. Because of the limitations of the stepper motor controller, this value must not be more than 832 mm above the home location.

step value of Z — this is the step distance between each point of data collection. Once a valid set of data has been collected at a given location, the bench will be moved up by this distance.

When the program has received the above information, it will automatically begin collection. Once finished, the main menu will be displayed and a new selection made.

A2.1.3 Program Hot Keys

While COLLECT is collecting data, there are several keys which can be pressed to affect the program actions. One of these keys can be pressed at any time and remain in effect until such time as another key is pressed. The keys are available in both intensity and range data collection.

I Do not repeat the data collection at the current point. If the data collection currently being done is valid, it will be saved. The bench will then move to the next location. This is useful if there is a problem with the current scan.

P Plot the current scan on the screen. This plot can be used as a manual method of checking the scan collection. In intensity data collection, the raw intensity data are plotted along with the locations of the targets derived from it. For the range collection, only the range data are plotted.

X Exit from the program at the end of this collection. This can be used as an emergency escape.

A2.2 Manual Validation of Intensity Data

To assist in the validation of the intensity data, a program called VALIDATE, which could plot a display of all the target bars locations for each scan file, was developed on the PC. This is a useful tool since it easily shows any scans which contain invalid target locations. An example of the output of the program is shown in Fig. A3.

Running the program is straightforward. It requests the range of the Z distances to be displayed and the name of the data file which is to be used. The data file is the file created in the intensity data collection called X_Z.PTS. The program will then plot the points contained in that file.

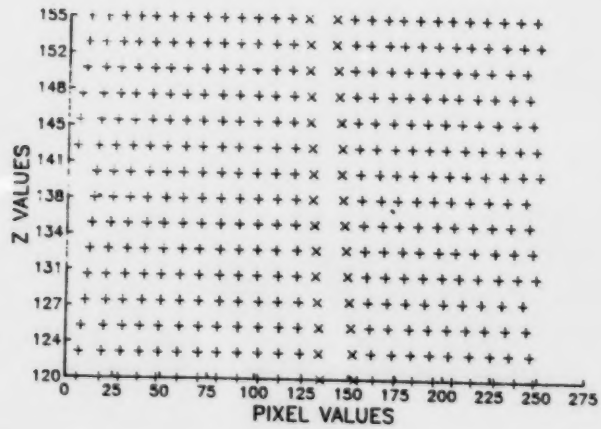

Figure A3. A sample plot produced by VALIDATE. The detected target positions are used to calculate the *X* equations.

REFERENCES

[A1] CY512/KIT For Evaluation and Prototyping, Stepper Motor Controller Manual. Cybernetic Micro Systems, P.O. Box 3000, San Gregorio, CA.

[A2] GM Counter GML-T Series 164 Operation Manual. Mitutoyo Mfg. Co. Ltd. 33-7, Shiba 5-chrome, Minato-ku, Tokyo, Japan.

APPENDIX B

Software used to control the calibration bench.

```
/* File: bench.c
 * Path:c:\scanner\calibrat\BENCH.C
 * Author:  Brian Trethewey
 * Date:    March 1988.
 * Purpose: Talk to the bench interactively
 *          or have the  bench move to a location automatically.
 */

#include <litecomm.h>     /*  header for comm software  */
        /*  see LC.DOC       */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>

/*  the program uses COM1 for talking to the bench       */
/*  for some reason, COM3 did not work         */

#define PORT 1

/*  WARNINGS:                */
/*  The characters returned from the bench sometimes come with  */
/*  bit 7 set; this was fixed by chopping off bit 7.  I do not  */
/*  know why this is happening, but it seems to work as it is   */
/*  used here.               */

main(c,v)
int c;
char *v[];
{
  int i,ch,stage,step;
  long loc;
  char buffer[80],*p;
  float current,final,last,base;

  /*  open the communications port         */
```

```
comm_opn(PORT,B1200,NPARITY,BIT8,STOP1,1024,1024);

/*  if there are not exactly two arguments, assume interactive*/

      if(c!=3) {

  /*  show that we are in interactive mode    */

        puts("You are now talking directly to the bench");
        puts("Type ESC to exit");

  /*  loop until someone hits escape      */

        while(1) {

    /*  work on keyboard input      */

          if(kbhit()) {

      /*  upper case please...    */

            ch=toupper(getch());

      /*  escape ends interactive and   */
      /*  quits program      */

            if(ch=='\033') {
        comm_close(PORT);
        exit(0);
      }

      /*  filter CR to CRLF on console  */

            if(ch=='\r')putchar('\n');
            else putchar(ch);

      /*  put character to bench    */

            lc_put(PORT,ch);
          }

    /*  work on characters from bench    */

          if(-1!=(ch=lc_get(PORT))) {

      /*  print character to console  */
      /*  filter CR to CRLF    */

            ch&=0x7f;
            if(ch=='\r')putchar('\n');
```

```
            else putchar(ch);
          }
        }
      }

/*  argument should be location and home location offset  */

      final=atof(v[1]);
      base=atof(v[2]);

/*  calculate location in bench coordinates      */

      loc=(long)(((final-base)/25.4)*2000L);

/*  stage 1 : goes to the location calculated    */
/*  stage 2 : do single steps until past requested position  */
/*  stage 3 : go to closest bench position to request  */

      stage=1;
      do {

  /*  set up the movement command      */

  sprintf(buffer,"P %ld\rB\rC\r",loc);

  /*  send command to bench        */

  for(p=buffer;*p!='\0';lc_put(PORT,*(p++)));

  /*  keep asking the bench where it is until an  */
  /*  answer is returned        */

  while(lc_icnt(PORT)<5) {
    if(kbhit())getch();
    lc_put(PORT,'B');
    lc_put(PORT,'\r');
    lc_put(PORT,'C');
    lc_put(PORT,'\r');
    for(i=10000;i>0;i--);
  }


  /*  get the answer          */

  for(p=buffer;;p++) {
    do {
      ch=lc_get(PORT);
      if(kbhit())getch();
      for(i=10000;i>0;i--);
    } while(ch<0);
```

```
    ch&=0x7f;
    if(ch=='\r')break;
    *p=ch;
  }
  *p='\0';

  /*  convert the answer to floating point    */

  current=10.0*atof(buffer+1);

  /*  figure out what to do next      */

  switch(stage) {

    /*  we are only in the rough area of the  */
    /*  position requested, so move toward the  */
    /*  position        */

    case 1:
      if((current-final)>0.0) step=-1;
      else step=1;
      last=current;
      stage=2;
      break;

    /*  we moved past the requested position,  */
    /*  so check to see when we were closest  */
    /*  and move to there      */

    case 2:
      if(((current-final)*step)>=0.0) {
        if(fabs(current-final)<=fabs(last-final))
          stage=4;
        else {
          stage=3;
          step*=-1;
        }
      } else {
        last=current;
      }
      break;

    /*  we are as close as we can get    */

    case 3:
      stage=4;
      break;
  }

  /*  set the new calculated position      */
```

```
    loc+=step;
  } while(stage<4);
  comm_close(PORT);
}



/*
 * File: movement.c
 * Path: c:\scanner\calibrat\collect
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose:  Allows the bench to be move to absolute locations according
to
 *           the parameters passed.
 */

#include <stdio.h>
#include <process.h>

float offset;     /*  the offset of the home location */

/*  have the user initialize the bench using BENCH.EXE */

void bench_setup()
{
  puts("Please set the home location using the 'A' command");
  puts("below the lowest location you will be moving to.");
  puts("For instance, if you are planning to move to 100 mm,");
  puts("move the bench to 98 mm and type 'A'.  The home location");
  puts("must be at over 1 mm below the lowest point to which you");
  puts("will be moving.\n");
  puts("Going into interactive mode with bench...");
  system("bench");
  puts("The offset displayed on the meter should now be entered.");
  printf("What offset is to be used in talking to the bench : ");
  scanf("%f",&offset);
}

/*  move the bench to the new z point using BENCH.EXE  */

void bench_move(loc)
float loc;
{
  char buffer[80];

  if((loc-offset)<1.0) {
    printf("Illegal bench position %f (%f)\n",loc,loc-offset);
```

```
    return;
  }
  sprintf(buffer,"bench %f %f",loc,offset);
  system(buffer);
}
```

APPENDIX C

Software used to collect and verify raw data for the calibration.

```
/*
 * File: collect.h
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 *  and the averaging of range values, and interpolation of X values.
 */

#define PIXELS 255          /* number of pixels read in a scan */
#define TARGETS 100         /* the maximum number of targets in single
scan */
#define SEEK_SET 0          /* set input file to first record */
#define SEEK_START 0L       /* set input file to first byte offset */
#define SATURATION 255.0    /* saturation of intensity values */



/*
 * File: collect.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: This mainline controls the collection of data for the
calibration
 *  and the averaging of range values, and interpolation of X values.
 */

#include <math.h>
#include <stdlib.h>
#include <malloc.h>
#include <stdio.h>
#include "color.h"
#include "collect.h"


/*  allocate an array of floats  */

float *array_init() {
```

```
  float *p;

  p=(float *)calloc(PIXELS,sizeof(float));
  if(p==NULL) {
    puts("Out of memory");
    exit(-1);
  }
  return(p);
}

main()
{
  int i, window, answer, ch,psize,option,key;
  float deltx;
  float junk_array[PIXELS];
  float *finten,*frange,*fival;
  float *new_int,*pix_val,*pix_dif,*x_pop;
  float *deriv1,*deriv2;
  float *x_val;
  float z_begin,z_end,z_step,z;
  float minx,maxx,minz,maxz;
  char buffer[80];
  FILE *mani;
  FILE *fout = NULL;

  /* initialize */
  finten=array_init();
  frange=array_init();
  fival=array_init();
  new_int=array_init();
  pix_val=array_init();
  pix_dif=array_init();
  x_pop=array_init();
  deriv1=array_init();
  deriv2=array_init();
  x_val=array_init();

  /* obtain the data */
  puts("********* Data Collection For Scanner Calibration *********\n");
  puts("Initialize the calibration bench as documented in BENCH.DOC.\n");
  bench_setup();

  gets(buffer);

  mani=fopen("manifest","w");
  if(mani==NULL) {
    puts("Could not open manifest file");
    exit(-1);
  }
```

```
/*  keep looping until you want to leave  */

while(1) {

  /*  wait for a valid selection  */

  do {
    printf("Type the letter of the operation you ");
    puts("would like to perform\n");
    puts("I. Intensity data collection");
    puts("R. Range data collection");
    puts("X. Exit");
    printf("\nOption : ");
    option=toupper(getche());
    printf("\n");
    if(option=='X') exit(0);
  } while(!((option=='I')||(option=='R')));

  key=' ';

  /*  wait for a valid low value of z  */

  for(z_begin=0.0;z_begin==0.0;) {
    printf("What is the starting position : ");
    gets(buffer);
    z_begin=atof(buffer);
  }

  /*  wait for a valid high value of z  */

  for(z_end=0.0;z_end==0.0;) {
    printf("What is the ending position : ");
    gets(buffer);
    z_end=atof(buffer);
  }

  /*  wait for a valid step size for z  */

  for(z_step=0.0;z_step==0.0;) {
    printf("What is the movement step : ");
    gets(buffer);
    z_step=atof(buffer);
  }

  /*  if intensity collection was requested  */

  if(option=='I') {

    /*  check to see what the spacing between targets is */
```

```
for(deltx=0.0;deltx==0.0;) {
  printf("What is the spacing of the black targets : ");
  gets(buffer);
  deltx=atof(buffer);
}

/*  open the support files  */
/*  manifest contains the ascii representation of all the */
/*  z values at which data is collected  */

/*  x_z.pts contains the z value, the position of the gap, */
/*  the number of points and the values of the points  */

fout=fopen("x_z.pts","w");
if(fout==NULL) {
  puts("Could not open verify file");
  exit(-1);
}

/*  for each of the z points, collect and manipulate data  */

for(z=z_begin;z<=z_end;) {
  printf("For Z = %f, ",z);

  /*  position the bench  */

  bench_move(z);

  /*  keep repeating until a good scan is acheived */

  while(1) {

    /*  check for a command key  */

    while(kbhit()) key=toupper(getch());

    /*  initialize the arrays  */

    for(i=0; i<PIXELS; i++)
      junk_array[i] =
        finten[i] = 0.0;

    /*  collect the data  */

    printf("Collecting...");
    collect_data(junk_array,finten);

    printf("Computing...");

    /* running average raw data */
```

```
run_ave(finten,new_int,50);

/* first and second derivatives of data */

deriv_scan(new_int,deriv1);
deriv_scan(deriv1,deriv2);

/* interpolate between maximum and minimums to find */
/* the zero crossing */

 psize =  min_max_int(deriv1,deriv2,
  pix_val);

/* perform the differance on the pixel values */

xdif(psize,pix_val,pix_dif);

/* *******   perform the X coordinate value assignment */

delx( lgap(psize,pix_dif),psize,
  pix_val,x_val,deltx);
while(kbhit()) key=toupper(getch());

/*  if the command key requests an exit */

if(key=='X') exit(0);

/*  if the command key requests a plot */

if(key=='P') {
  printf("Plotting...");
  for(i=0; i<PIXELS; i++)
    fival[i] = (float) i;
  bgnplot(1,'g',"d:junk");
  font(1,"simplex.fnt",'\310');
  startplot(8);
  page(9.0,6.855);
  area2d(8.0,5.5);
  upright(1);
  graf("%-1.0f",0.0,32.0,255.0,"%-1.0f",
    -10.0,32.0,255.0);
  color(RED);
  curve(fival,finten,PIXELS,0);
/*   finedot();
  curve(fival,new_int,PIXELS,0);
  color(BLUE);
  solid();
  curve(fival,deriv1,PIXELS,0);
  solid();
```

```
        color(MAGENTA);
        curve(fival,deriv2,PIXELS,0);
*/        color(YELLOW);
        for(i=0; i<psize; i++)
        {
          if(pix_val[i] < 500.0)
          {
            symbol(pix_val[i],
              0.0,3);
          }
        }
        endplot();
        stopplot();
      }

      /*  validate the intensity scan */

      if(vintensity(pix_dif,psize,lgap(psize,pix_dif))==0) {

        /*  invalid, so loop again */

        printf("BAD\n");
        printf("                    ");
      } else {

        /*  valid, so write it and leave loop */

        /*  save to the x_z.pts file */

        fprintf(fout,"%f %d %d ",z,
          lgap(psize,pix_dif),
          psize);
        for(i=0;i<psize;i++) {
          fprintf(fout,"%d ",
            (int)pix_val[i]
            );
        }
        fprintf(fout,"\n");

        /*  save z to the manifest file */

        fprintf(mani,"%f\n",z);

        /*  finish with the scan and write it */

        xpop(z,psize,x_pop,pix_val,
          x_val);
        printf("Done\n");
        break;
      }
```

```
        /*  if command key requests ignore, leave loop */

        if(key=='I') break;
      }

      /*  next value of z */

      z+=z_step;
    }

    /*  all done, so close files */

    fclose(mani);
    fclose(fout);
  }

  /*  if range data collection was requested */

  if(option=='R') {

    /*  for each point in z */

    for(z=z_begin;z<=z_end;) {
      printf("For Z = %f, ",z);

          /*  save z to the manifest file */

          fprintf(mani,"%f\n",z);

      /*  position the bench */

      bench_move(z);

      /*  loop until there is a valid scan */

      while(1) {

        /*  check for a command key */

        while(kbhit()) key=toupper(getch());

        /*  initialize the scan */

        for(i=0; i<PIXELS; i++)  frange[i] = junk_array[i] = 0.0;

        /*  collect the range scan */

        printf("Collecting...");
        collect_data(frange,junk_array);
```

```
/*  check for another command key */

while(kbhit()) key=toupper(getch());

/*  if an exit was requested */

if(key=='X') exit(0);

/*  if a plot was requested */

if(key=='P') {
  minz=frange[128]-256.0;
  maxz=frange[128]+256.0;
  minx=0;
  maxx=255;
  printf("Plotting...");
  for(i=0; i<PIXELS; i++) fival[i] = (float) i;
  bgnplot(1,'g',"d:junk");
  font(1,"simplex.fnt",'\310');
  startplot(8);
  page(9.0,6.855);
  area2d(8.0,5.5);
  upright(1);
  graf("%-1.0f",minx,32.0,maxx,"%-1.0f",
    minz,100.0,maxz);
  color(GREEN);
  curve(fival,frange,PIXELS,0);
  endplot();
  stopplot();
}

/*  validate the range */

if(vrange(frange)==0) {

  /*  range invalid, so loop again */

  printf("BAD\n                    ");
} else {

  /*  range valid, so write it and leave loop */

  printf("Done\n");
  write_disk((long)(z*1000),frange,"rop");
  break;
}

/*  if ignore requested, leave loop */
```

```
            if(key=='I') break;
          }

          /*  next z */

          z+=z_step;
        }
      }
    }
  }
```

```
/* File: collect1.c
 * Path: c:\collect1.c
 * Author: Doug Taylor....Colin Archibald
 * Purpose: Collects raw data from the scanner and averages 10 successive
 *          scans.
 * Note: This is used in conjunction with the program calib.hex on the
 *       profile scanner's 68000.
 */

#include "collect.h"

#define HEAD_A 0x000
#define HEAD_B 0x400
#define MY_HEAD HEAD_A
#define N_SCANS 10
#define MODE2           0xc2  /* Set Mode 2 (194) */
#define SET_PORT        0x28c /* Mode Port (652) */
#define STATUS_PORT     0x288  /* Status Port Address */
#define LEAST_SIG_PORT 0x284
#define MOST_SIG_PORT  0x280

short get_scanner();

void collect_data(ran_result, inten_result)
float ran_result[], inten_result[];
{
   /*
    * This program is intended to capture data from the scanner
    * using the C routine get_scanner. The data is averaged for
calibration.
    */
  short i,j;
  double temp_result[PIXELS];
```

```
  /* set status port on Lab-40  */
  outp(SET_PORT,MODE2);

  for(i=0; i<PIXELS; i++)
    temp_result[i] = 0;

  for(i=0;i<N_SCANS;i++) {
    for (j=0;j<PIXELS;j++) {
      while(((get_scanner()&0x1400)!=0x1400));
      temp_result[j] += (double) (get_scanner() & 0x0fff);
      inten_result[j] += (float) (get_scanner() & 0x0ff);
    }
  }   /* end for loop to collect data */

  /* average the scans and store them in frange, and finten
                                          ... global to calibration */
  for(i=0; i<PIXELS; i++)
  {
    ran_result[i] = (float) temp_result[i] / N_SCANS;
    inten_result[i] /= (float) N_SCANS;

  }
}

short get_scanner()
{

  union {
    short reading;
    char read[2];
  } in;

  while(!(0x20&inp(STATUS_PORT)));
  in.read[0] = inp(LEAST_SIG_PORT);
  in.read[1] = inp(MOST_SIG_PORT);
  return(in.reading);
}
```

```
/*
 * File:runxave.c
 * Path: c:\scanner\calibrat\collect
 * Author: Doug Taylor.
 * Date: March 10, 1988.
 * Purpose:  Perform a running average on the data moving the low pixels
 *           to saturation in an attempt to magnify the dips in the data
 *
 */

#include "collect.h"

void run_ave(old,new,run)
  float *old, *new;     /* the raw data */  /* new running average data */
  int run;             /* the running length of the data to be averaged */
  {

  int i,j,run_2;
  float ave;

  run_2=run/2;

  for(i=2;i<run;i+=2)
  {
    ave=0.0;
    for(j=0;j<=i;j++) ave+=old[j];
    ave/=(i+1);
    new[i/2]=SATURATION/ave*old[i/2];
  }

  for(i=run_2;i<PIXELS-run_2;i++)
  {
    ave=0.0;
    for(j=-run_2;j<=run_2;j++) ave+=old[i+j];

    /* count zero crossover when j=0 */
    ave/=(float)run+1;
    new[i]=SATURATION/ave*old[i];
  }

  run -=1;

  /* i maintains the value it left off at */
  for(;i<PIXELS-1;i++)
  {
    ave = 0.0;
    run_2=run/2;

    for(j=-run_2;j<=run_2;j++) ave+=old[i+j];
```

```
    ave/=(run+1);
    new[i] = SATURATION/ave*old[i];

    /* decrease the run by 2 each loop */
    run-=2;
  }
}



/*
 * deriv.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Take the derivative of the average intensity
 *          profile collected from the scanner during calibration.
 *
 */

#include "collect.h"

void deriv_scan(point,epoint)
float *point, *epoint;
{
  int i,j, op;
  float int1[PIXELS + 2];

  op = 2;

  for(i=0; i<PIXELS+op; i++)
     *(int1+i) = 0.0;

  /* input the INT values into array */
  for(i=op; i<PIXELS+op; i++) *(int1+i)=*(point+(i-op));

  /* take derivative by using -1 -1 0 1 1 algorithm */

  for(i=0;i<PIXELS-op;i++)
    *(epoint+i) = -1 * *(int1 + i) +
                  -1 * *(int1 + i + 1) +
                   1 * *(int1 + i + 3) +
                  1 * *(int1 + i + 4);

}
```

```
/*
 * File: minxmaxx.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Interpolates pixel positions and range positions at
 *          zero crossings.
 *
 */

#include "collect.h"

#define SAT_THRESH 200.0  /*  threshold for point selection */

int min_max_int(tempif,fordif,pix_val)
float *tempif,*fordif,*pix_val;
{
  int i,j,flag;

  for (j=0; j< TARGETS ; j++)
  {
    for (i=0; i< (PIXELS-1); i++)
    {
      if (((*(tempif+i)<0.0) && (*(tempif+(i+1))>0.0)) ||
        ( *(tempif+i) == 0.0) )
      {
        if (*(fordif+i)> SAT_THRESH)
        {
          /* first sign change */
          if ( *(tempif+i) == 0.0)
          {
            *(pix_val+j) = (float)i;
            j++;
            flag=j;
          }
          else
          {
            *(pix_val+j) = (*(tempif+i)/(*(tempif+i)-*(tempif+i+1)))+
               (float)i;
            j++ ;
            flag=j;
          }
        }
      }
    }
    break;
```

```
  }
  return ( flag );
}
```

```
/*
 * File: xdif.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Find the difference between each x point.
 *
 */

void xdif( size, pix_val, pix_dif )
int size;
float *pix_val,*pix_dif;
{
  int i;

  for ( i=0; i< size-1 ; i++)
    *(pix_dif+i)= *(pix_val+(i+1)) - *(pix_val+i);
}
```

```
/*
 * File: delx.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Accepts the spacing between the x targets and
 *          assigns a co-ordinate system to the pixel values.
 *
 */
void delx( post, size, pix_val, x_val, deltx )
int post,size;
float *pix_val,*x_val;
float deltx;
{
  int i,j;

  j=1; /* initializing counter */

  for (i=post; i > -1 ; i--)
```

```
  {
    if ( *(pix_val+i) != 0.0 )
    {
       *(x_val+i) = (-1) * ( deltx ) * j;
       j++;
    }
    else
      printf(" pixel value is zero X value not assigned ");
  }
  j=1; /* initializing counter */

  for (i=post+1; i<size; i++)
  {
    if ( *(pix_val+i) != 0.0 )
    {
      *(x_val+i) =  ( deltx ) * j;
      j++;
    }
    else
      printf(" pixel value is zero X value not assigned\n ");
  }
}



/*
 * File: lgap.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Finds the largest gap in the x data.
 *.
 */

int lgap( size, pix_dif )
int size;
float *pix_dif;
{
  int i;
  int post;  /* position of largest gap */
  float max; /* max. differance */

  max = 0.0;

  for(i=0 ; i<size ; i++)
  {
    /* assump. there is only one wide gap detected */
    if ( *(pix_dif+i)> max )
    {
```

```
      max = *(pix_dif+i);
      post = i;
    }
  }
  return(post);
}
```

```
/*
 * File:xpop.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Interpolates using pixel values, x values for 1 scan line.
 *
 */

#include "collect.h"

void xpop( z, size, x_pop, pix_val, x_val )
float z;
int size;
float *x_pop,*pix_val,*x_val;
{
  int i,j, con1, con2;  /* counters for pixel values */
  char *ext1;

  /* intializing the X value array */

  for (j=0; j<PIXELS ;j++) *(x_pop+j) = 0.0;

  /* interpolating in the X value array  */

  for ( i=0; i< size-1 ; i++)
  {
    con1=(int) (pix_val[i]+1.0);
    con2=(int) pix_val[i+1];

    while( con1 <= con2 && con2 >0 )
    {
      x_pop[con1] = x_val[i] * ( (con1 - pix_val[i+1])/(pix_val[i] -
                    pix_val[i+1]) );

      x_pop[con1] += x_val[i+1] * ( (con1 - pix_val[i])/( pix_val[i+1]
                    - pix_val[i]) );
```

```
      con1++;
    }
  }

  /* extrapolating on the left side of the scan (- ve) */

  for ( i=(int)pix_val[0]; i>-1 ;i--)
    x_pop[i]= x_pop[i+1] - x_pop[i+2] + x_pop[i+1];

  /* extrapolating on the right side of the scan ( +ve) */

  for ( i= con2+1; i< PIXELS ;i++)
    x_pop[i]= x_pop[i-1] - x_pop[i-2] + x_pop[i-1];

  write_disk ( (long)(z*1000.0), x_pop, "xop" );
}



/*
 * File:  writexdi.c
 * Path: c:\scanner\calibrat\collect
 * Author: Shahram Amid
 * Date: March 10, 1988.
 * Purpose: Writes the files to the disk to be used in later calibration.
 *
 */

#include <stdio.h>
#include "collect.h"

void write_disk (num, array1, ext1 )
long num;
float *array1;
char *ext1;
{
  FILE *file_point;
  char file_1[20];
  int i;

  sprintf(file_1,"%8.8ld.%s",num,ext1);
  file_point= fopen(file_1,"wb");

  if(file_point==0)
  {
    printf("unable to open file %s",file_1);
    exit(0);
  }
```

```
  /* write to disk */
  fwrite(array1,PIXELS,sizeof(float),file_point);
  fclose(file_point);
}
```

```
/* File: verify.c
 * Path: c:\scanner\calibrat\collect
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose: Verify that the range scan is reasonably valid
 *  this validation is based on there not being any sudden changes
 *  in the data of the scan.
 */

#include <stdio.h>
#include <math.h>
#include "collect.h"

int vrange(r)
float *r;
{
  int i;
  float max_diff,t;

  max_diff=0.0;

  /*  ignore the ends of the scan, since there is usually a problem there
*/

  for(i=30;i<(PIXELS-50);i++) {
    t=fabs(*(r+i)-*(r+i+1));
    if(t>max_diff) max_diff=t;
  }

  /*  was there a sudden change? */

  if(max_diff>30) return(0);
  else return(1);
}

/*  the percentage difference between the first value and the second */

float perc(f1,f2)
float f1,f2;
{
```

```
  return((f2/f1)-1.0);
}

/*  check the intensity scan for validity */
/*  based upon there being no abnormal changes in the space sizes */

int vintensity(diffs,siz,gap)
float *diffs;  /*  differences between spaces */
int siz,gap;  /*  number of points, and where the largest gap is */
{
  int i;    /*  working variable  */
  float t;  /*  current difference */
  float max_perc;  /*  largest difference */

  /*  check that the gap is in a reasonable location */

  if((siz-gap)<1) return(0);
  if(gap<1) return(0);

  /*  look for the biggest space before the gap */

  max_perc=0.0;
  for(i=1;i<gap;i++) {
    t=perc(*(diffs+i-1),*(diffs+i));
    if(t>max_perc) max_perc=t;
  }

  /*  check the gap itself, which is twice the normal space */

  t=perc(*(diffs+gap-1),*(diffs+gap)/2.0);
  if(t>max_perc) max_perc=t;
  t=perc(*(diffs+gap)/2.0,*(diffs+gap+1));
  if(t>max_perc) max_perc=t;

  /*  check the rest of the spaces */

  for(i=gap+2;i<(siz-1);i++) {
    t=perc(*(diffs+i-1),*(diffs+i));
    if(t>max_perc) max_perc=t;
  }
  printf("Quality(%.3f)...",max_perc);

  /*  were there any abnormal spaces? */

  if(max_perc > .6) return(0);
  else return(1);
}
```

```
/* File: validate.c
 * Path: c:\scanner\calibrat\collect
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose: This program reads in the x_z.pts file created by the
intensity
 *  data collection part of COLLECT.EXE and displays it in a manner
 *  which should make it easy to tell if there is a problem with any
 *  of the data points.  The result should look like a set of radiating
 *  lines of points down from the top of the graph.  There should not
 *  be any points missing in any of these lines (although there may be
 *  some missing at the edges of the graph).
 */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "color.h"

main()
{
  float zup,zlo,x,z;
  int i,gap,n;
  char name[80],buffer[1024];
  FILE *fin;

  printf("Z lower bound of the data set : ");
  gets(buffer);
  zlo=atof(buffer);
  printf("Z upper bound of the data set : ");
  gets(buffer);
  zup=atof(buffer);
  printf("XZ-pts file name : ");
  gets(name);
  if(NULL==(fin=fopen(name,"r"))) {
    puts("Could not open input file");
    exit(-1);
  }
  bgnplot(1,'g',"junk");
  font(1,"simplex.fnt",'\310');
  startplot(8);
  page(9.0,6.855);
  area2d(8.0,5.5);
  upright(1);
  color(WHITE);
  graf("%-1.0f",0.0,25.0,255.0,"%-1.0f",zlo,(zup-zlo)/10.0,zup);
  color(RED);
  xname("PIXEL VALUES");
  yname("Z VALUES");
```

```
    color(BLUE);
    heading("P-Z Bar Locations");
    color(GREEN);
    while(1)
    {
      if(EOF==fscanf(fin," %f %i %i",&z,&gap,&n)) break;
      for(i=0;i<gap;i++) {
        fscanf(fin," %f",&x);
        symbol(x,z,3);
      }
      for(;(i<(gap+2))&&(i<n);i++) {
        fscanf(fin," %f",&x);
        symbol(x,z,4);
      }
      for(;i<n;i++) {
        fscanf(fin," %f",&x);
        symbol(x,z,3);
      }
    }
    endplot();
    stopplot();
}
```

APPENDIX D

Software used to calculate the *X* equations from the raw data.

```
/* File: xeqns.c
 * Path: c:\scanner\calibrat\xcal
 * Author: Colin Archibald
 * Date: March 1988
 * Purpose: The purpose here is compute the linear equations for the
azimuth
 * contours.  The X values for several different Z's have been located
 * in COLLECT.  They are in a data set file which has a manifest file
 * indicating what the z values are.  We read in all the data, and then
 * use a least squares fit algorithm to determine the eqn of each
azimuth.
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <malloc.h>
#include <ctype.h>

#define PIXELS 255

/*  the current data path  */

char directory[80];
double huge xvals[50][PIXELS], zvals[50], azxvals[50];

main()
{
  FILE *fp_zvl, *fname;      /*  manifest file containing all Z values */
  /*  the parameters file name, and a working buffer space  */
  char buffer[80];
  int ans,t,i,con1,con2,j,k;  /*  various working variables  */
  double m,b;

  /*  get the drive and directory of the first data set */

  printf("\n\nCompute the equations for the azimuths.\n\n");
  printf("\nWhere is the data (drive:directory) : ");
```

```
 gets(directory);

 /*  open the manifest file.. for x there is only 1 of these */

 sprintf(buffer,"%s\\manifest",directory);
 if(NULL==(fp_zvl = fopen(buffer,"r"))) {
   printf("Could not open %s\n",buffer);
   exit(1);
 }

 /*  get the first data file */

 fscanf(fp_zvl," %lf",zvals);
 getx_data(zvals[0], xvals);

 /* move the x values to the local origin of the camera */
 for(i=0; i<PIXELS; i++)
   xvals[0][i] += 10000.0;

 /*  repeat until done */
 k = 0;
 while ( 1 ) {

     /*  get the next value of z from the manifest */
     k++;
     if(1!=fscanf(fp_zvl," %lf",&zvals[k])) {
       printf("Finished reading data\n");
       break;
     }

   /*  display the current value of z, and get the data file */

   printf("Current z = %lf      \r",zvals[k]);
   getx_data(zvals[k],&xvals[k]);

   /* move the x values to the local origin of the camera */
    for(i=0; i<PIXELS; i++)
     xvals[k][i] += 10000.0;

 } /* end data reading  there are k different z's */


 sprintf(buffer,"%s\\xeqn",directory);
 if(NULL==(fname = fopen(buffer,"w"))) {
   printf("Could not open %s\n",buffer);
   exit(1);
 }

   /*  for each azimuth, compute the m and b parameters */
```

```
    for ( i=0; i<PIXELS ; i++)
    {
      for(j=0; j<k; j++)
        azxvals[j] = xvals[j][i];

      linearls(azxvals, zvals, k, &m, &b);
      fprintf(fname,"%d. %f %f\n",i,m,b);

    }
}


/* File: linfit.c
 * Path: c:\scanner\calibrat\xcal
 * Author: Colin Archibald
 * Date: July 1988
 * Purpose: Fit a straight line to the set of x-z points.  There are
 *          n points, thef equation z=mx+b is solved in a least squares
 *          sense, and the m and b are returned.
 */

void linearls(x, z, n, m, b)
double huge x[], z[];
int n;
double *m, *b;
{
  double sigx, sigz, sigx2, sigxz;
  int i;

  /*
  sigx  sum of all x values
  sigz  sum of all z values
  sigx2 sum of all x values squared
  sigxz sum of all x*z values
  */

  sigx = sigz = sigxz = sigx2 = 0;

  for(i=0; i<n; i++)
  {
    sigx += x[i];
    sigz += z[i];
    sigxz += x[i] * z[i];
    sigx2 += x[i] * x[i];
  }

  *m = (n * sigxz - sigx * sigz) /  ((n*sigx2) - (sigx * sigx));
  *b = (sigz - *m * sigx) /  n;

}
```

```
/* File: getx_dat.c
 * Path: c:\scanner\calibrat\xcal
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose: Fetch the raw X target data that was collected in the data
 *          in the previous stage.  Use the naming convention to allow
 *          the user to easily access the raw data.
 */

#include <stdio.h>
#define PIXELS 255

extern char directory[];  /*  the current data path  */

void getx_data(z,xvalue)
double z;          /*  the current z position */
double *xvalue;        /*  the buffer for the x data */
{
  char filenam[80];
  FILE *fp;
  int i;
  float buffer[PIXELS];

  /*  construct the filename from the directory and z value */

  sprintf(filenam,"%s\\%08.8ld.xop",directory,(long) (1000.00 * z));
  if(NULL==(fp = fopen(filenam,"rb"))) {
    printf("Could not open %s.\n",filenam);
    exit(-1);
  }

  /*  read in the array of data */

  fread(buffer,sizeof(float),PIXELS,fp);
  for(i=0;i<PIXELS;i++) {
    xvalue[i]=(double)buffer[i];
  }
  fclose(fp);
}
```

APPENDIX E

Software used in the generation of the Z lookup table. The interpolation of the raw data is in this section.

```
/* File: zlookup.c
 * Path: c:\scanner\calibrat\lookup
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose: Create the Z lookup table, using linear interpolation.  The
 *          user is asked to specify where the raw data is located, and
 *          the manifest files are read to determine the names of the
 *          raw data files.
 */

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <malloc.h>
#include <ctype.h>

#define PIXELS 255

/*  the lookup table structure  */

struct table {
  int huge *buffer;
  long low,high,start;
  FILE *fptr;
} *tab;

/*  the current data path  */

char directory[80];

/*  routines for entering and getting data from the lookup tables  */

int  get_table(struct table *,long);
void put_table(struct table *,long,int);
struct table *init_table(char *);
struct table *load_table(char *);

/*  interpolate a given z table cell between two sets of scan data */
```

```
double zinterp(con,zl,zh,pzl,pzh)
int con;
double zl,zh,pzl,pzh;
{
  return((((con-pzh)/(pzl-pzh))*zl)+(((con-pzl)/(pzh-pzl))*zh));
}

main()
{
  double *pzh;          /*  high set of range data  */
  double *pzl;          /*  low set of range data  */
  double zl,zh;        /*  positions of the high and low scan sets  */
  FILE *fp_zvl;        /*  file containing all valid z values in this data
set*/

  char *fname,buffer[80];  /*  the table name, and a working buffer space
*/
  int ans,t,i,con1,con2,j;  /*  various working variables  */

  /*  allocate storage for the range and xvalue arrays */

  if((NULL==(pzh=(double *)calloc(PIXELS,sizeof(double))))||
     (NULL==(pzl=(double *)calloc(PIXELS,sizeof(double))))){
    puts("Memory error");
    exit(-1);
  }

  fname = "lut.z";

  /*  if requested, */
  /*  recreate the file, initialized to zeroes */
  /*  else, simple read in the file */

  printf("\nReinitialize existing table (y/n) ? ");
  ans=getche();
  if(toupper(ans) == 'Y') tab=init_table(fname);
  else tab=load_table(fname);

  /*  get the drive and directory of the first data set */

  printf("\nWhere is the data (drive:directory) : ");
  gets(directory);

  /*  open the first manifest file */
  sprintf(buffer,"%s\\manifest",directory);
  if(NULL==(fp_zvl = fopen(buffer,"r"))) {
    printf("Could not open %s\n",buffer);
    exit(1);
  }
```

```
/*  get the first data file */

fscanf(fp_zvl," %lf",&zl);
getz_data(zl,pzl);

/*  repeat until done */

while ( 1 ) {

  /*  if there are no more data files in that manifest file */

  if(1!=fscanf(fp_zvl," %lf",&zh)) {

    /*  close the old file and ask for a new one */

    fclose(fp_zvl);
    while( 1 ) {
      puts("\nIf you have more data, enter where it can be found,");
      printf("otherwise just press return : ");
      gets(directory);

      /*  if none are given, then everything is done */

      if(directory[0]=='\0') {

        /* fill trailing cells with constant value ( 1 ) */

        for(i=0;i<PIXELS;i++) {
          printf("Finishing %d    \r",i);
          for(j=2047;0==get_table(tab,i+(j*(long)PIXELS));j--) {
            put_table(tab,i+(j*(long)PIXELS),1);
          }
        }
        save_table(tab);
        exit(0);
      }

      /*  otherwise try to open it, */
      /*  if it can't be opened, ask again */

      sprintf(buffer,"%s\\manifest",directory);
      if(NULL==(fp_zvl = fopen(buffer,"r"))) {
        printf("Could not open %s\n",buffer);
        continue;
      } else break;
    }

    /*  get the next value of z from the manifest */
```

```
      if(1!=fscanf(fp_zvl," %lf",&zh)) {
        puts("Invalid Manifest file");
        exit(-1);
      }
    }

    /*  display the current value of z, and get the data file */

    printf("Current z = %lf      \r",zh);
    getz_data(zh,pzh);

    /*  for each element in the data file */

    for ( i=0; i<PIXELS ; i++) {

      /*  find the bounded area between data files */

      con1 = (int)((*(pzl+i)+1.0)/2.0);
      con2 = (int)((*(pzh+i)+1.0)/2.0);
      if( con1 > con2 ) {
        j=con1;
        con1=con2;
        con2=j;
      }

      /*  fill in the entire bounded area */

      for(;con1<con2;con1++) {

      /*  fill in the space dependant on which table is */
      /*  being worked on */

      put_table(tab,i+(con1*(long)PIXELS),
        (int)((10*zinterp(con1*2,zl,zh,pzl[i],pzh[i]))+.5));
      }
    }

    /*  copy the current file to the old file */

    memcpy(pzl,pzh,sizeof(double)*PIXELS);

    /*  change lower z to the higher z and restart the loop */

    zl=zh;
  }
}
```

```
/* File: getz_dat.c
 * Path: c:\scanner\calibrat\lookup
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose: Fetch the raw range data that was collected in the data
 *          in the previous stage.  Use the naming convention to
 *          allow the user to easily access the raw data.
 */

#include <stdio.h>
#define PIXELS 255

extern char directory[];  /*  the current data path  */

void getz_data(z,range)
double z;          /*  the current z position */
double *range;        /*  the buffer for the range data  */
{
  char filenam[80];
  FILE *fp;
  int i;
  float buffer[PIXELS];

  /*  construct the filename from the directory and z value */

  sprintf(filenam,"%s\\%08.8ld.rop",directory,(long) (1000.00 * z));
  if(NULL==(fp = fopen(filenam,"rb"))) {
    printf("Could not open %s.\n",filenam);
    exit(-1);
  }

  /*  read in the array of data */

  fread(buffer,sizeof(float),PIXELS,fp);
  for(i=0;i<PIXELS;i++) {
    range[i]=(double)buffer[i];
  }
  fclose(fp);
}
```

```
/* File: vmem.c
 * Path: c:\scanner\calibrat\lookup
 * Author: Brian Trethewey
 * Date: March 1988
 * Purpose: A virtual memory handler for creating files that are
 *          larger than the 640 K available on the PC
 */

#include <stdlib.h>
#include <malloc.h>
#include <stdio.h>

#define V_BLOCK 255
#define N_BLOCKS 900

struct table {
  int huge *buffer;
  long low,high,start;
  FILE *fptr;
};

/*  read in as much of the table as is possible */

struct table *load_table(name)
char *name;
{
  struct table *temp;
  int i;

  /*  create the table structure */

  if(NULL==(temp=(struct table *)malloc(sizeof(struct table)))) {
    puts("Memory error");
    exit(-1);
  }

  /*  allocate the table buffer space */

  if(NULL==(temp->buffer=
        (int huge *)halloc(V_BLOCK*(long)N_BLOCKS,sizeof(int)))) {
    puts("Memory error");
    exit(-1);
  }

  /*  open the table file */

  if(NULL==(temp->fptr=fopen(name,"r+b"))) {
    puts("File open error");
    exit(-1);
```

```
  }

  /*  read in the file, with a status indicator */

  printf("\n");
  for(i=0;i<N_BLOCKS;i++) {
    if(!(i&31)) printf("Reading %d\r",i);
    if(V_BLOCK!=
      fread(temp->buffer+(i*V_BLOCK),sizeof(int),V_BLOCK,temp->fptr)) {
      puts("Read error");
      exit(-1);
    }
  }
  printf("Reading %d\r",i);

  /*  set the lower and upper bounds of in memory table, and */
  /*  offset in the buffer of the start of the table */

  temp->low=0L;
  temp->high=V_BLOCK*(long)N_BLOCKS;
  temp->start=0L;
  return(temp);
}

/*  write a file containing zeros as the initialized table */
/*  and then call load_table to read it in */

struct table *init_table(name)
char *name;
{
  FILE *fptr;
  int line[V_BLOCK],i;

  /*  open the file */

  if(NULL==(fptr=fopen(name,"wb"))) {
    puts("File open error");
    exit(-1);
  }

  /*  set a line of zeroes */

  for(i=0;i<V_BLOCK;i++) {
    line[i]=0;
  }

  /*  write the table */

  printf("\n");
  for(i=0;i<2048;i++) {
```

```
    if(!(i&31)) printf("Setting %d\r",i);
    if(V_BLOCK!=fwrite(line,sizeof(int),V_BLOCK,fptr)) {
      puts("Write error");
      exit(-1);
    }
  }
  printf("Setting %d\r",i);
  fclose(fptr);

  /*  now load the new table */

  return(load_table(name));
}

/*  shift the table so that the requested address is in memory */

void shift_table(t,addr)
struct table *t;
long addr;
{

  /*  keep shifting lower in the table until the address is in */

  while(addr<t->low) {

    /*  set the new start offset and upper and lower bounds */

    if(t->start==0L) t->start=(N_BLOCKS-1)*(long)V_BLOCK;
    else t->start-=(long)V_BLOCK;
    t->low-=(long)V_BLOCK;
    t->high-=(long)V_BLOCK;

    /*  seek to the proper position in the table and write the overflow
*/

    if(0!=fseek(t->fptr,sizeof(int)*t->high,SEEK_SET)) {
      puts("Seek error");
      exit(-1);
    }
    if(V_BLOCK!=fwrite(t->buffer+t->start,sizeof(int),V_BLOCK,t->fptr)) {
      puts("Write error");
      exit(-1);
    }

    /*  seek to the proper position and read in the new block */

    if(0!=fseek(t->fptr,sizeof(int)*t->low,SEEK_SET)) {
      puts("Seek error");
      exit(-1);
    }
```

```
    if(V_BLOCK!=fread(t->buffer+t->start,sizeof(int),V_BLOCK,t->fptr)) {
      puts("Read error");
      exit(-1);
    }
  }

  /*  keep shifting higher in the table until the address is in */

  while(addr>=t->high) {

    /*  seek to the proper location and write the overflow */

    if(0!=fseek(t->fptr,sizeof(int)*t->low,SEEK_SET)) {
      puts("Seek error");
      exit(-1);
    }
    if(V_BLOCK!=fwrite(t->buffer+t->start,sizeof(int),V_BLOCK,t->fptr)) {
      puts("Write error");
      exit(-1);
    }

    /*  seek to the proper location and read the new block */

    if(0!=fseek(t->fptr,sizeof(int)*t->high,SEEK_SET)) {
      puts("Seek error");
      exit(-1);
    }
    if(V_BLOCK!=fread(t->buffer+t->start,sizeof(int),V_BLOCK,t->fptr)) {
      puts("Read error");
      exit(-1);
    }

    /*  set the new start offset and upper and lower bounds */

    if(t->start==((N_BLOCKS-1)*(long)V_BLOCK)) t->start=0L;
    else t->start+=(long)V_BLOCK;
    t->low+=(long)V_BLOCK;
    t->high+=(long)V_BLOCK;
  }
}

/*  return the value at the requested address in the table */

int get_table(t,addr)
struct table *t;
long addr;
{
  long offset;

  /*  get the address in memory */
```

```
  shift_table(t,addr);

  /*  calculate the real address and return the value */

  offset=(t->start+(addr-t->low))%(V_BLOCK*(long)N_BLOCKS);
  return(*(t->buffer+offset));
}

/*  put a value at the requested location in the table */

void put_table(t,addr,v)
struct table *t;
long addr;
int v;
{
  long offset;

  /*  get the address in memory */

  shift_table(t,addr);

  /*  calculate the real address and store the value in it */

  offset=(t->start+(addr-t->low))%(V_BLOCK*(long)N_BLOCKS);
  *(t->buffer+offset)=v;
}

/*  save the in memory portion of the table and deallocate all the
storage */

void save_table(t)
struct table *t;
{
  int i;
  long offset;

  /*  for each of the memory blocks */

  for(i=0;i<N_BLOCKS;i++) {

    /*  seek to the proper location and save it */

    if(!(i&31)) printf("Writing %d    \r",i);
    if(fseek(t->fptr,sizeof(int)*(t->low+(V_BLOCK*(long)i)),SEEK_SET)) {
      puts("Seek error");
      exit(-1);
    }
    offset=(t->start+(V_BLOCK*(long)i))%(V_BLOCK*(long)N_BLOCKS);
    if(V_BLOCK!=
```

```
        fwrite(t->buffer+offset,sizeof(int),V_BLOCK,t->fptr)) {
        puts("Write error");
        exit(-1);
      }
    }

    /*  free the table buffer, close the table file and */
    /*  free the table structure */

    hfree(t->buffer);
    fclose(t->fptr);
    free(t);
}



/* File: lookztab.c
 * Path: c:\scanner\calibrat\lookup
 * Author: Colin Archibald
 * Date: March 1988
 * Purpose: Plots a sampling of lines of the luts for verification of
 *          roughly correct LUT data.
 */

#include <stdio.h>
#include "color.h"

#define PIXELS 255    /* number of pixels giving information (0,255) */


float x[PIXELS];
float calz[PIXELS]; /* the calibrated range data from the luts */
int  lutz[PIXELS];
int i,j,lines;
FILE *fopen(), *zlut;

main()
{
  char zinfile[30];
  int n,step_size,start;

  for(i=0; i<PIXELS; i++)  x[i] = (float) i;

  printf("Enter the z lut infile name:\n");
  gets(zinfile);
  printf("%s\n", zinfile);

  /* load the lookup tables */
  zlut = fopen(zinfile,"rb");
```

```
    printf("How many lines in file:(1024 or 2048?)\n");
    scanf("%d",&lines);

    printf("Start plotting at:\n");
    scanf("%d",&start);
    printf("Step through file step size:");
    scanf("%d", &step_size);


    bgnplot(1,'g',"lutz.tkf");
    font(1,"simplex.fnt", '\310');
    startplot(BLACK);
    page(9.0,6.855);
    area2d(8.0,5.5);
    upright(1);
    graf("%-1.0f",0.0,32.0, 256.0, "%-1.0f", 0.0, 1000.0, 11000.0);
    yname("\310 Z Table Values ");
    heading("\310Look at Z LUT");
    color(RED);
    /* look at every step_size lines of the tables */
    for(i=0; i<lines; i++)
    {
      n=0;
      fread((char *)lutz, sizeof(short int), PIXELS, zlut);
      color(2);
      if(((i % step_size ) == 0) && (i > start))
      {
        for(j=0; j<PIXELS; j++)
        calz[j] = (float) lutz[j];
        curve(x,calz,PIXELS,0);
      }
    }
    endplot();
    stopplot();
}
```